Отчётпо лабораторной работе №9

Дисциплина: Архитектура компьютера

Максимова Дарья Валерьевна

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программам с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

По аналогии с предыдущими лабораторными работами, файлы asm я буду создавать в каталоге “work/arch-pc”. В файле lab9-1.asm вписываю программу из листинга 9.1, которую я внимательно изучила. Создаю исполняемый файл и запускаю его в работу. (рис. 1).

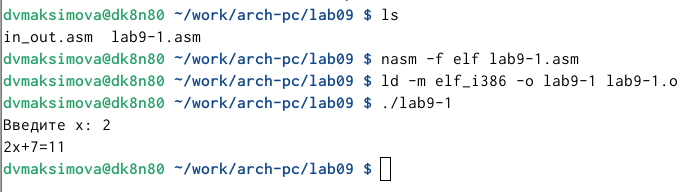


Рис. 1: Первые действия в лабораторной работе и результат программы

После этого я переписываю программу согласно методическому материалу (рис. 2).

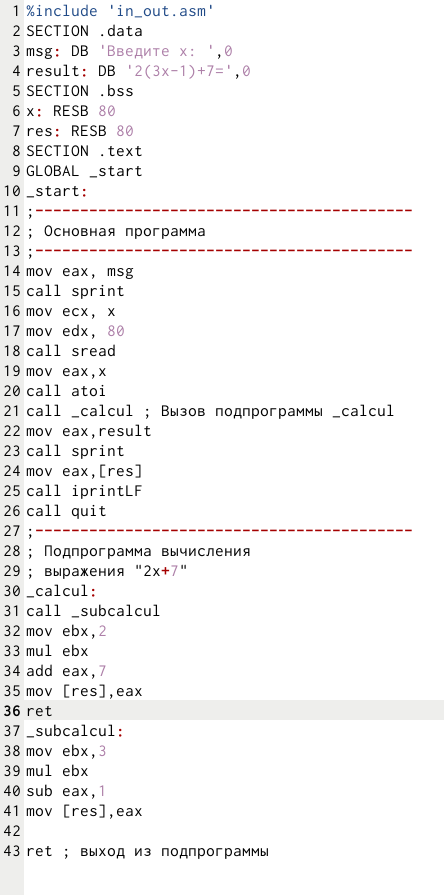


Рис. 2: Текст программы

Создаю исполняемый файл и убеждаюсь в том, что все работает успешно.(рис. 3).

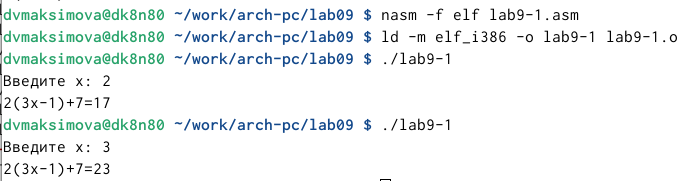


Рис. 3: Результат программы

## 3.2 Отладка программам с помощью GDB

В новом файле lab9-2.asm записываю текст программы из листинга 9.2. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ провожу с ключом ‘-g’. И загружаю файл в откладчик gbd, а затем проверяю выполнение программы с помощью команды run (рис. 4).

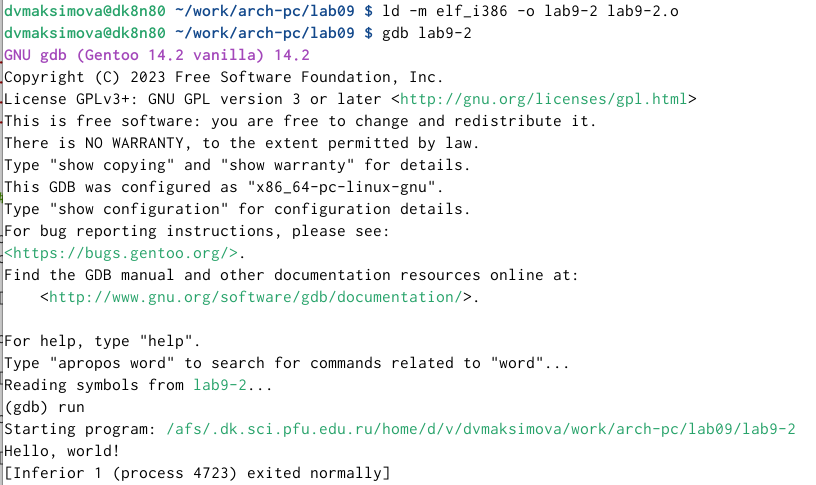


Рис. 4: Работа программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. 5).

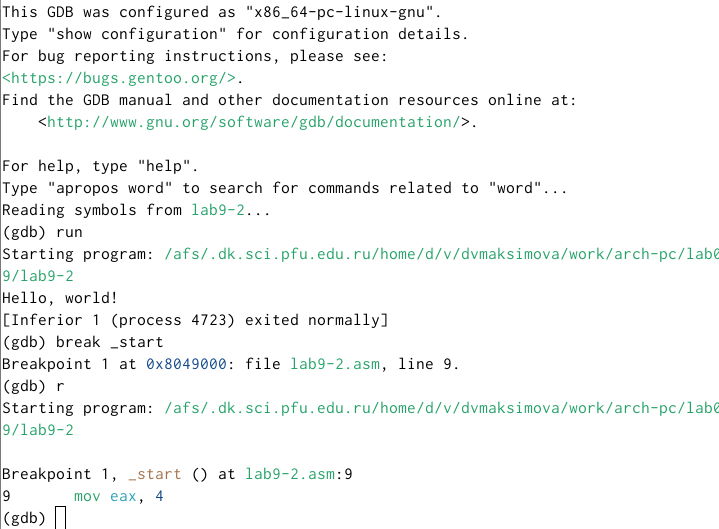


Рис. 5: Брейкпоинт

Просматриваю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 6).



Рис. 6: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 7).



Рис. 7: Отображение команд

Есть некоторые различия в отображениях в этих режимах, а именно в виде колонки с текстом программы: в Intel’e она выглядит, как *“$0x{операнд},%{регистер}”*, а в ATT - *“{регистер},0х{операнд}”*

Включаю режим псевдографики для удобного анализа программ (рис. 8).



Рис. 8: Псевдографика.Режим

## 3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (рис. 9).

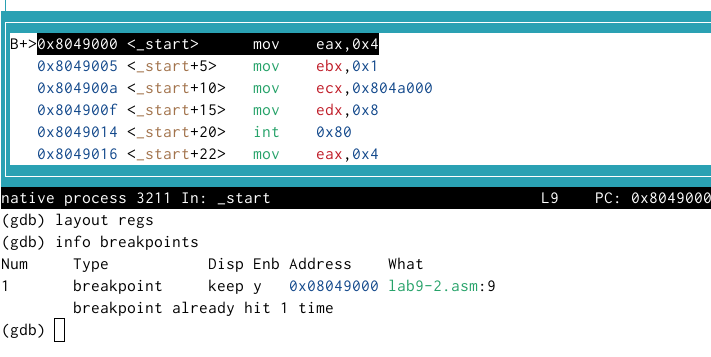


Рис. 9: Информация о точке остановы

Я установила еще одну точку останова по адресу инструкции, и теперь когда мы запрашиваем информацию о брейкпоинтах, нам показываются две точки останова: (рис. 10).

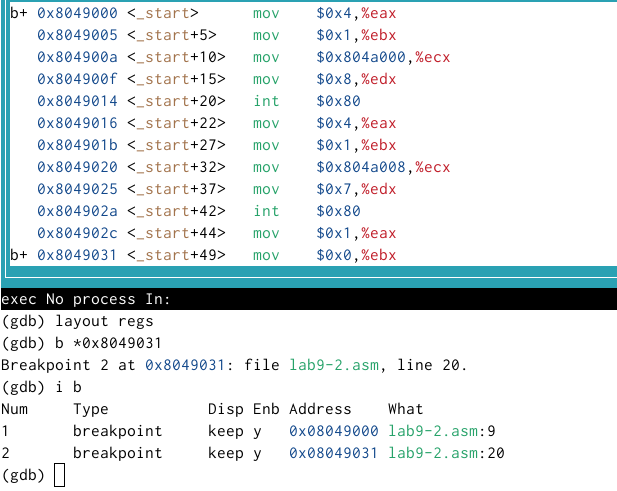


Рис. 10: Точка останова

## 3.4 Работа с данными программы в GDB

Для того, что бы посмотреть значения регистров можно воспользоваться командой “i r”. Вот какие значения регистров вывелись (рис. 11).

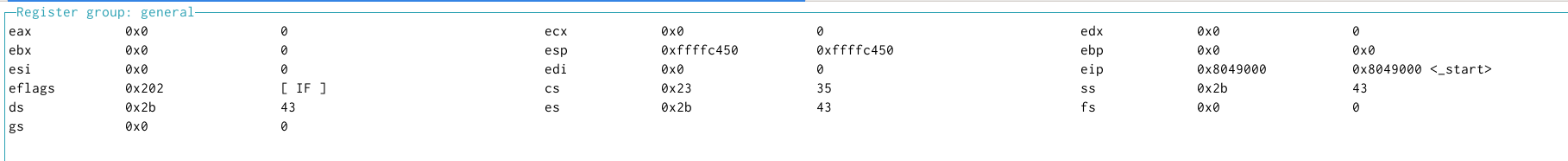


Рис. 11: Значения регистров

С помощью команды x & также можно посмотреть содержимое переменной. Я посмотрела значение переменной msg1 по имени и вот результат. (рис. 12).

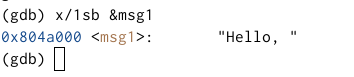


Рис. 12: Просмотр значения переменной

Аналогично можно посмотреть значение переменной msg2 по адресу. (рис. 13).

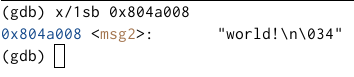


Рис. 13: Значение другой переменной

Теперь я изменяю первый символ переменной msg1 (рис. 14).

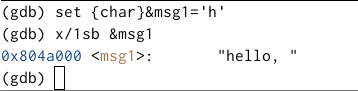


Рис. 14: msg1=hello,

А также я изменила разные символы в переменной msg2 (рис. 15).

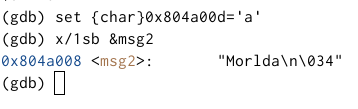


Рис. 15: Измененные символы

Теперь изменим значение регистра ebx: (рис. 16).

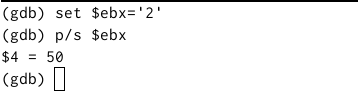


Рис. 16: Изменение ebx

Команда set позволяе изменить значение регистра, проделываем это с регистром ebx, и использую команду p/s, чтоб промотреть значение.

## 3.5 Обработка аргументов командной строки в GDB

Создаю исполняемый файл из файла, с которым я работала в процессе выполнения лбораторной работы №8.

А затем загружаю данный файл в отладчик, предварительно указав ключ к работе с аргументами, и указываю эти самые аргументы (рис. 17).

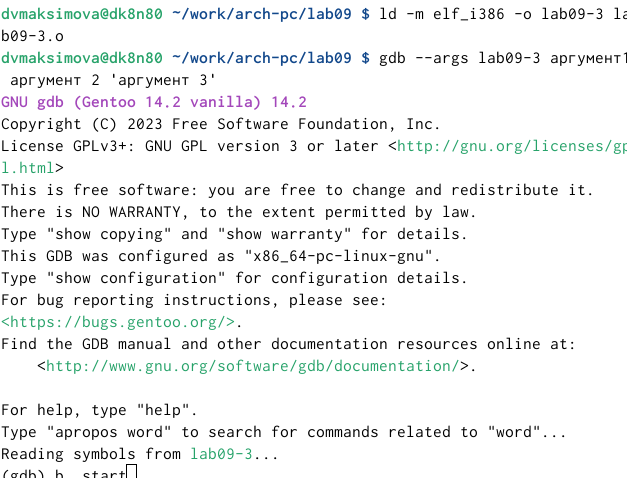


Рис. 17: Переход к отладчику

Для начала устанавливаем первую точку останова, а затем запускаем программу (рис. 18).

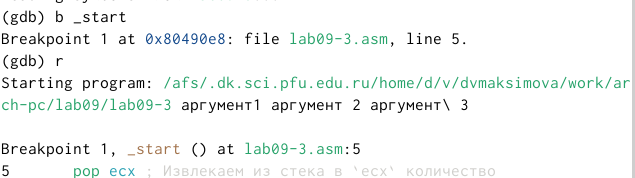


Рис. 18: b-point + run

На данном этапе, мы можем просмотреть позиции стека отдельно по адресам (рис. 19).

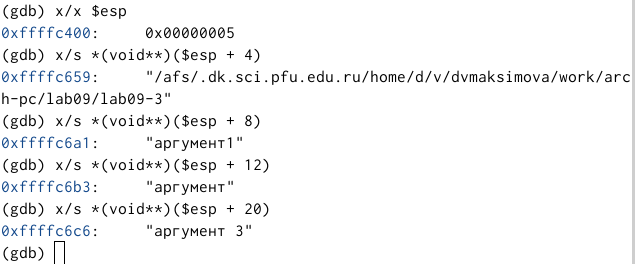


Рис. 19: Позиции стека

## 3.6 Задание для самостоятельной работы

Переходим к выполнению заданий самостоятельного характера.

№1

Я преобразовываю программу , с которой я работала в процесее 8 лабораторной работы. Я должна реализовать вычисление функции в подпрограмме (до этого вычисления были в основной программе).

Вот таким образом выглядит код программы: (рис. 20).

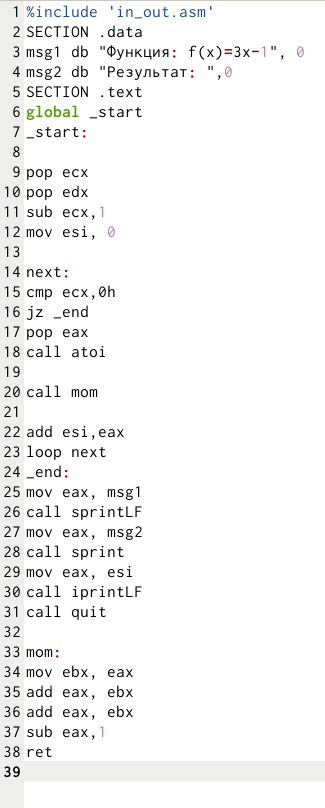


Рис. 20: Код программы

Теперь проверим как работает программа: (рис. 21).

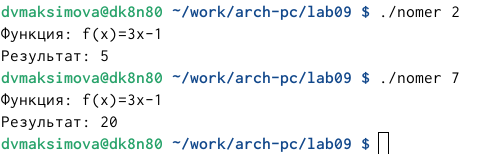


Рис. 21: Результат

№2

При запуске программа вычисления выражения (3 + 2) ∗ 4 + 5 дает неверный результат. Программа должна выводить 25, а выводит в ответе 10.(рис. 22).

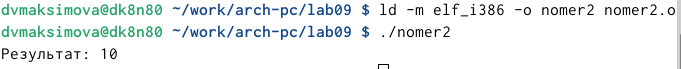


Рис. 22: Ложный результат(символьный)

Моя задача - определить ошибку с помощью отладчика. Ошибка заключается в том, что регистр eax был умножен на 4, хотя для правильности нам надо было умножать ebx

Саомстоятельно постараюсь исправить код. Вот так выглядит код с внесением моих изменений (рис. 23).

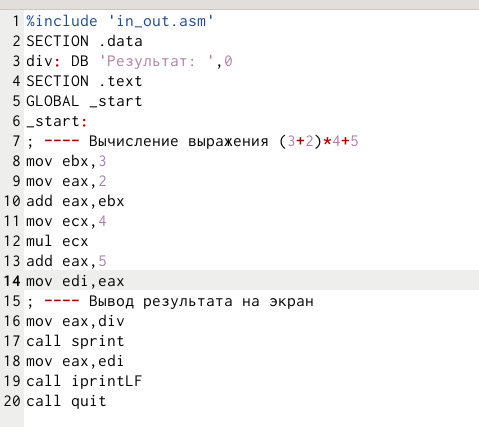


Рис. 23: Код

Теперь я проверяю работу кода, и радуюсь, что все нормально работает!!!:) (рис. 24).

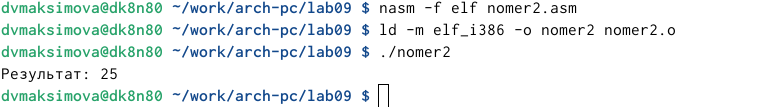


Рис. 24: Результат

# 4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм, познакомилась с методами отладки при помощи GDB и его основными возможностями.